Running Java on GPUs

An ESA Innovation Triangle Initiative with:

- **ESA's Gaia mission**
- **Observatoire de Genève**
- **Ateji**

All Gaia code is in Java

Introducing OpenCL or CUDA in the development process would be a huge disruption

Ateji PX embeds parallelism in Java

Multi-core parallelism already available, GPU target developed during this project

→Port compute-intensive period search algorithms to the GPU

Deeming, Lomb-Scargle, String-Length

How it works:

- **Source-to-source translation from fragments of Java to OpenCL**
- **Computation locality and communications expressed with Ateji PX extensions**

**1. Express parallelism with || **

```
for(Obs o : observations) ...
```

Becomes

```
|| (Obs o : observations) ...
```

► *Computation now uses all available cores.*

2. Express distribution to GPU with # annotations

```
|| (#OpenCL(), Obs o : observs)
   // these branches run on GPU
|| (...)
   // these branches run on CPU
```

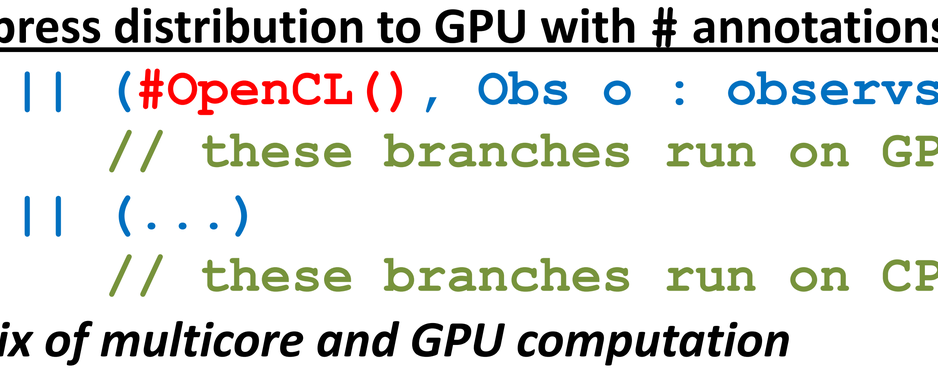
► *Mix of multicore and GPU computation*

3. Express communication with message passing

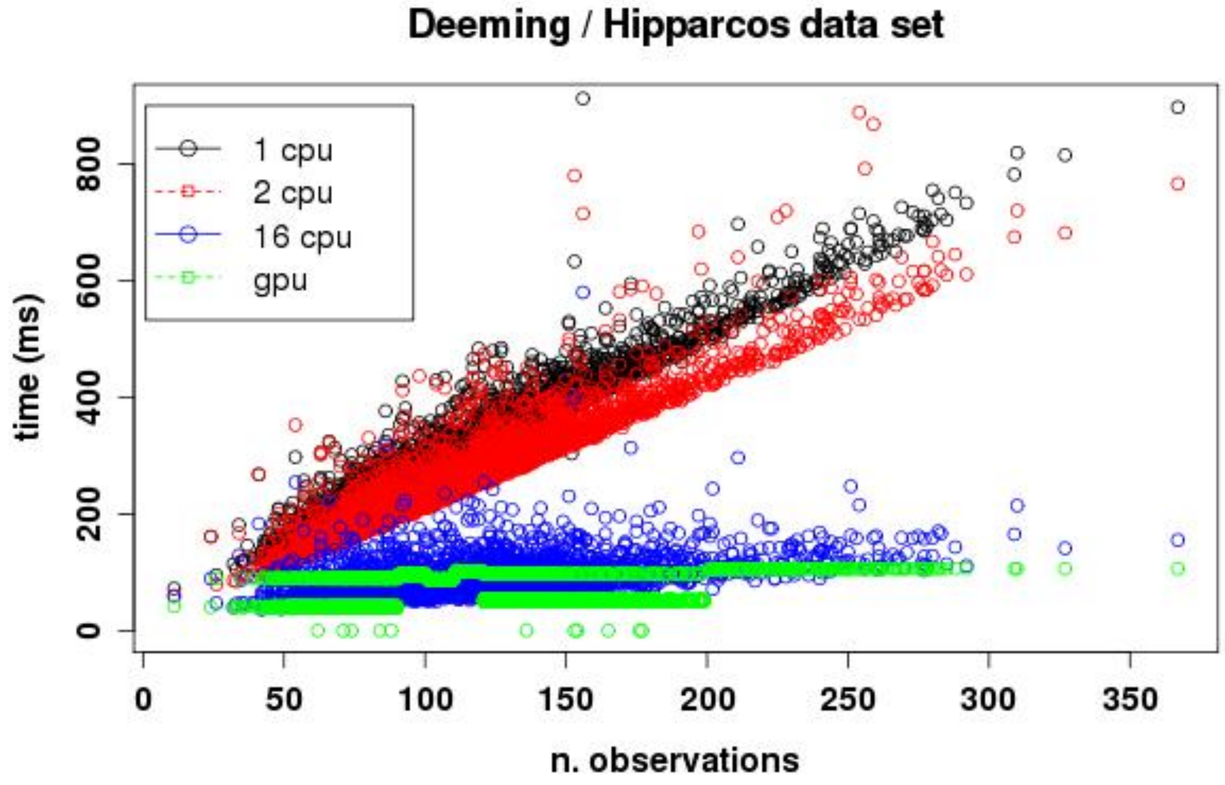
```
Channel c = ...;
[
  || c ! value;
  || c ? value;
]
```

► *Overlap of computation and communication*

4. Map the GPU memory hierarchy to lexical scope



Benchmarks:



Lessons learned:

- **Parallelism easy to introduce in the code**
  - **No disruption in the Java devt. process**
  - **Pretty good speed-up on multi-cores**
- **Similar perf. figures for GPU and 16-core GPU**
  - **But depend heavily on the hardware**
  - **Still a lot of room for improvement**
- **Code needs refactoring for performance on GPU, such as loop pipelining and workgroup sizing**
  - **Even though we code in Java, detailed knowledge of GPU architecture is required**
- **Performance / price ratio on multi-core vs. CPU**
  - **Not decisive**
  - **But hardware is evolving rapidly**

Contacts: mathias.beck@unige.ch, patrick.viry@ateji.com

Downloads & Documentation: www.ateji.com/px